SANDIA REPORT

SAND2017-12477
Unlimited Release
Printed November 2017

Simulation Data Management – Requirements and Design Specification

Robert L. Clay, Ernest J. Friedman-Hill, Marcus J. Gibson, Edward L. Hoffman, Daniel Laney, Kevin H. Olson

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology and Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from

U.S. Department of Energy

Office of Scientific and Technical Information

P.O. Box 62

Oak Ridge, TN 37831

Telephone: (865) 576-8401

Facsimile: (865) 576-5728

E-Mail: reports@osti.gov

Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce

National Technical Information Service

5301 Shawnee Rd

Alexandria, VA 22312

Telephone: (800) 553-6847

Facsimile: (703) 605-6900

E-Mail: orders@ntis.gov

Online order: http://www.ntis.gov/search

SAND2017-12477
Unlimited Release
Printed November 2017

Simulation Data Management – Requirements and Design Specification

Robert L. Clay, Ernest J. Friedman-Hill, Marcus J. Gibson, Edward L. Hoffman
Scalable Modeling and Analysis Department

Kevin H. Olson
SAIC

Daniel Laney
LLNL

Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185

Abstract

Simulation Data Management (SDM), the ability to securely organize, archive, and share analysis models and the artifacts used to create them, is a fundamental requirement for modern engineering analysis based on computational simulation. We have worked separately to provide secure, network SDM services to engineers and scientists at our respective laboratories for over a decade. We propose to leverage our experience and lessons learned to help develop and deploy a next-generation SDM service as part of a multi-laboratory team. This service will be portable across multiple sites and platforms, and will be accessible via a range of command-line tools and well-documented APIs. In this document, we'll review our high-level and low-level requirements for such a system, review one existing system, and briefly discuss our proposed implementation.

CONTENTS

FIGURES

1 INTRODUCTION

According to NAFEMS* Simulation Data Management (SDM) is the effective management of simulation data and process information to ensure support for digital product development, manufacturing, and life cycle management. SDM is a technology which uses database technology to enable users to manage structures of simulation and process data across the complete product lifecycle. SDM artifacts can be data, models, processes, documents and metadata relevant to modeling, simulation, and analysis. SDM provides the ability to securely organize, archive, and share analysis models and the artifacts used to create them. A good SDM system provides version control for documents being worked on, and archiving for completed projects. It also preserves and records the provenance of all the data is contains, so years afterward someone can reconstruct exactly what went into an experiment or calculation, and how a result was obtained.

In a general SDM system, spanning multiple engineering disciplines, the data is less structured than what is typically stored in a Product Data Management (PDM) system. There are multiple reasons for this reduced structure, though one primary reason is the lack of a controlling CAD file. On the other hand, SDM requires more structure than a classic Document Management System (DMS), as properties like analysis domain, data provenance, team membership and manager/employee relationships form an important part of how the data is stored, searched and retrieved.

In a secure environment based on need-to-know (NTK) information access, a mature security model is an important part of an SDM system's capabilities. The ability to integrate with existing access control mechanisms is a strict requirement, as is a security model with the granularity needed to express the dynamic and sometimes complex set of permissions afforded to the individual.

Large numbers of files, some of them very large in terms of data size, can be generated while creating and executing analysis models. An SDM system needs to be able to scale to handle these conditions, or have a mechanism for dealing with them. In addition, many analysis files are transitory in nature. While there is a need to control and share them during a limited time frame, these files may be safely discarded after an analysis project is complete.

Computational simulation (CompSim) is important to our several missions, and we are charged with providing SDM to support CompSim at our sites, some of us for over a decade. As CompSim continues to grow both in importance and in scale, our teams hope to jointly deploy a new and more powerful SDM service for a broader range of applications. It is hoped that by pooling resources, we can create a new service with enhanced capabilities and characteristics for a larger customer base. By defining a shared specification with an open-source implementation, we hope to be able to deploy a better system at a reduced cost, and provide value to a larger audience.

---

* https://www.nafems.org/publications/browse_buy/browse_by_topic/data/what_is_simulation_data_management_/

2 CONCEPT OF OPERATIONS

2.1 General definitions

In this chapter, we present a conceptual model for Simulation Data Management as understood in this report. Subsequent chapters will flesh out this model by enumerating concrete requirements and use cases, and present an example of an existing system which implements the model.

For the purposes of this report and the system we propose to build, the fundamental unit of data storage is the *file.* A file is an opaque sequence of bytes (or more properly, a collection of one or more *versions*, each of which is an opaque sequence of bytes); in particular the SDM system enforces no structure on the contents of a file. Although some features of the system (like full text search) may access the contents of some files, this does not imply that the SDM system knows anything about those contents or their format. This stands in contrast to a hypothetical system for storage of structured data in which the entire content of each datum is understood by the system; for instance, a system that explicitly stores tables of numbers and can retrieve specific values on demand, or even perform computations on them. Such a system obviously makes significant assumptions about the data that can be stored in it, and trades power for generality.

Files are the fundamental unit of data storage, but the *artifact* is the basic unit of organization. An artifact has a collection of *metadata* and contains zero or more files. The metadata include everything the system knows about an artifact. An artifact also has a *type*, which is effectively a schema for the metadata it is expected to have and a partial mapping for the contents of the metadata. Artifacts can therefore represent everything from a document or folder to an event like a simulation run.

An artifact is connected to other artifacts via *relationships*, which themselves are artifacts and therefore contain arbitrary metadata to describe the nature of the relationship. A relationship can represent anything from containment of a document in a folder, to a dependency between a simulation result and an input deck. Note that the metadata for the latter kind of a relationship can include details about the simulation process – analysis code, hardware platform, software versions – that was used to create the dependent file.

Artifacts that represent documents will be organized in *folders* and ultimately in *projects*. A project, with an associated *team* of users, is a primary mechanism for access (and access control.) Typically a user will work with a single project at a time. Sites that do not wish to organize work this way could simply use one project, with one all-encompassing team, to hold all of the organization's documents. Of course, a project has its own associated metadata; in particular, a project can belong to a specific *campaign*.

2.2 Basic Use Cases

Although an SDM system can be quite complex, the fundamental use cases are quite simple, and are listed here in this section. Later chapters of this document will fill in many details, but ultimately the following small set of use cases cover most of what the system is expected to do.

2.2.1 Commit artifacts

The client software allows users to commit organized collections of files to the repository. The files are stored securely at the server and the organization and metadata about the file are preserved.

2.2.2 Query artifacts

The client can perform queries to determine what artifacts are available, understand their organization, and list their metadata. The query capability should be sufficient to support a project explorer that allows users to graphically browse the contents of a project within the repository (Figure 1). The directory structure within the project folder is mirrored between the graphical client and server. Visual decorators allow users to easily identify which files and folders are local-only, remote only, or on both the client and server.

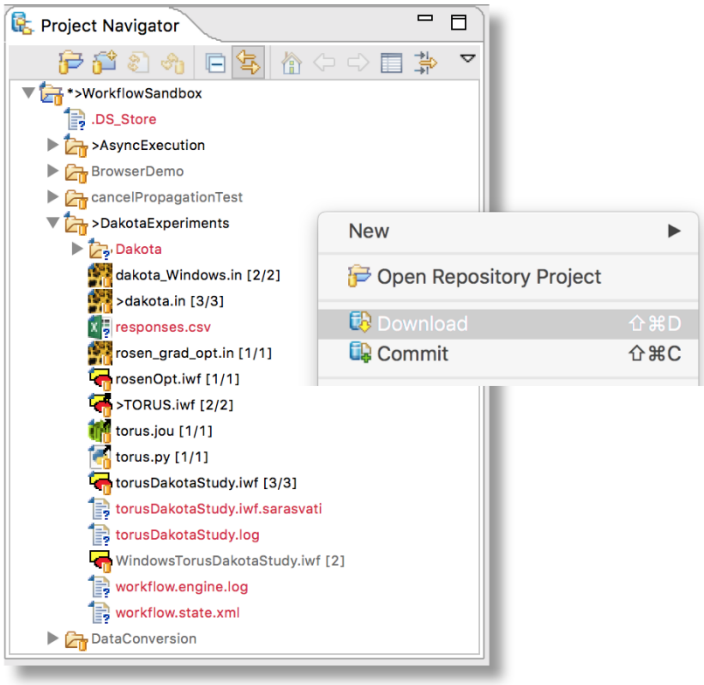

Figure 1. Project Navigator View

2.2.3 Download artifacts

If the user wants to view the contents of a given project file or edit it, they must download it to their local workspace. The user can select individual files to download, or they can select any folder, including the overall project folder, and recursively download the entire contents of that folder.

3 USER REQUIREMENTS

3.1 Configuration management

3.1.1 Document Versioning

The system shall allow users to commit version changes to documents. This will include the following version metadata:

3.1.1.1 Version owner

3.1.1.2 Revision comment

3.1.1.3 Revision date

*3.1.2 Baselining**

A baseline is a collection of documents that go together to form a higher-level object. For example, baselines are used in design engineering to tag part files that make up an assembly. In modeling and simulation, baselines can be used to keep track of the files associated with a model and its simulation results files. Models can be composed of multiple mesh files, and input file, include files, etc.

3.1.2.1 Baseline creation: The system shall allow users to create baselines from a collection of files.

3.1.2.2 Baseline downloading: The system shall allow users to download all files, and only those files, associated by a given baseline tag.

3.1.2.3 Edit baselines: Not sure if this is needed or even rigorous data management. The idea is that a user could add or remove files from an existing baseline.

3.1.3 Dependency Management

The system shall provide mechanisms to keep track of parent/child relationships between files. While this sounds similar to baselines, it is a fundamentally different concept. Baselines are collections of files that go together to represent a higher level thing. File dependencies provide provenance metadata as to how a file came to exist. Examples of dependency relationships include:

- A CAD model is a parent to a mesh created from it
- An input file and a mesh file are parents to a simulation result file

3.2 Security

3.2.1 Flexible authentication system

The SDM system shall be able to authenticate users via an organization's own systems. A Kerberos integration should detect an existing Kerberos ticket, should one exist, and use that to authenticate the user.

3.2.2 Personnel Database Interface

The SDM system should have an interface to industry standard personnel databases (e.g. LDAP).

3.2.3 Need to Know (NTK) Access Control

In general, access control is based on team membership and team roles per the SDM system. In addition, team management shall be provided by custom integrations with third party access control systems.

3.3 Document Access Control

The system shall organize all information into projects, which will associate a given set of files with a set of users ('the team members'). In general, documents checked into a project have common need to know access. In the case where team members import documents from other projects, the access control list of the originating project in used.

3.3.1 Project Browsing

Users shall have the ability to browse projects that they have access to. The browse operation should include obtaining a list of project contents and any subdirectories.

3.3.2 File name search

*3.3.3 Full text search**

3.3.4 Metadata based search

3.3.4.1 Document type

3.3.4.2 Owner

3.3.4.3 Document attributes. This could include user attributes applied to documents as well as the standard set of attributes (e.g. creation date, size, etc)

3.3.5 Search and Need-To-Know (NTK)

All searches can only return results that the user can see based on NTK controls. Results must be filtered on the server not on the client.

3.3.6 Document URL's

Documents in the SDM repository can be referenced via URL's that can be pasted in documents (e.g. Word or PowerPoint).

3.4 Document Metadata

3.4.1 Classification

Documents are marked with a classification that can be used to adjudicate NTK. This should be implemented in a generalized way. An individual site can use this to tag documents from a standard set of classification.

3.4.2 Engineering notes

The SDM system will provide mechanisms to attach note to documents.

3.4.3 Standard document attributes

The SDM system will have a standard set of document attributes, including file size, creation date, mod date, MD5 check sum, etc.

3.4.4 User attributes

The SDM system should allow users to define their own name/value attributes.

3.5 Interfaces

3.5.1 RPC interface

The primary API shall be an object-based remote procedure call interface, such as a web services interface. Other interfaces can be implemented atop this interface.

3.5.2 Command line interface

At the most basic level of functionality, a command line interface will allow users to check in and check out files from the SDM repository. The command line interface will provide access to other SDM functionality that is analogous to a GUI client.

3.5.3 Web Interface

A web interface will allow users to access documents via a web browser, without having to install client software. This interface will be useful for sharing information with content consumers (e.g. managers and the analyst's customers) more so than content providers (e.g. analysts)

3.5.4 Workflow component interface

The workflow interface will allow users to access the SDM repository (e.g. commit or download files) as a task in an automated workflow.

3.6 Project Requirements Management

The system shall allow users to assign a level of formality (LOF) to a project.

3.6.1 Level of Formality

3.6.2 Requirements

3.6.3 Peer Review

4 FUNCTIONAL REQUIREMENTS

In this section, we provide a full list of SDM-related system functions that the new service would be expected to support. Although the new service may not support each of these operations directly, it must be possible to compose each of these functions out of more fundamental operations.

4.1 Project Actions

Projects associate a given body of work within an hierarchical structure with a given set of team members. Projects are intended to have a specific lifetime, and a project should have an identifiable set of deliverables. Of course, projects may be used as a general storage area without the need for a time frame or specific outcome. By associating teams to projects, a project is a primary focus for access control.

4.1.1 Roles

A project has 1..*N* individuals who participate in the creation of artifacts in a project. Within a project, there are three roles:

4.1.1.1 Team Member : may create folders and artifacts within a Project.

4.1.1.2 Team Lead : in addition to being a Team Member, may do the following:

4.1.1.2.1 Modify the team membership. Adjusting the team is implicitly a change to Need-to-Know.

4.1.1.2.2 Make changes to the access control of a project

4.1.1.2.3 The team lead can take ownership

4.1.1.3 Project Manager: in addition to being a Team Lead, may also conduct certain operations pertaining to the lifecycle of the project.

In addition to the Project Roles, the concept of an *artifact owner* is important. The individual who originates an artifact is its owner. The owner of a piece of data has the ability to determine certain aspects of its sharing, and can restrict the visibility of an artifact.

4.1.2 Create Project

Any authenticated user may create a Project. Project creation collects certain attributes about the project. The user may specify a Project Manager who is different than the initiating user.

4.1.2.1 Sample Existing Project Attributes

- Project Name
- Description of the project. Ideally this description should indicate the expected deliverables
- Classification (UUR -> SRD)
- Classification Modifiers (Export Controlled, ITAR)

- The Analysis Discipline (e.g., thermal, structural, etc.; should be multi-select)
- Responsible Manager
- Project start, estimated finish, actual finish.

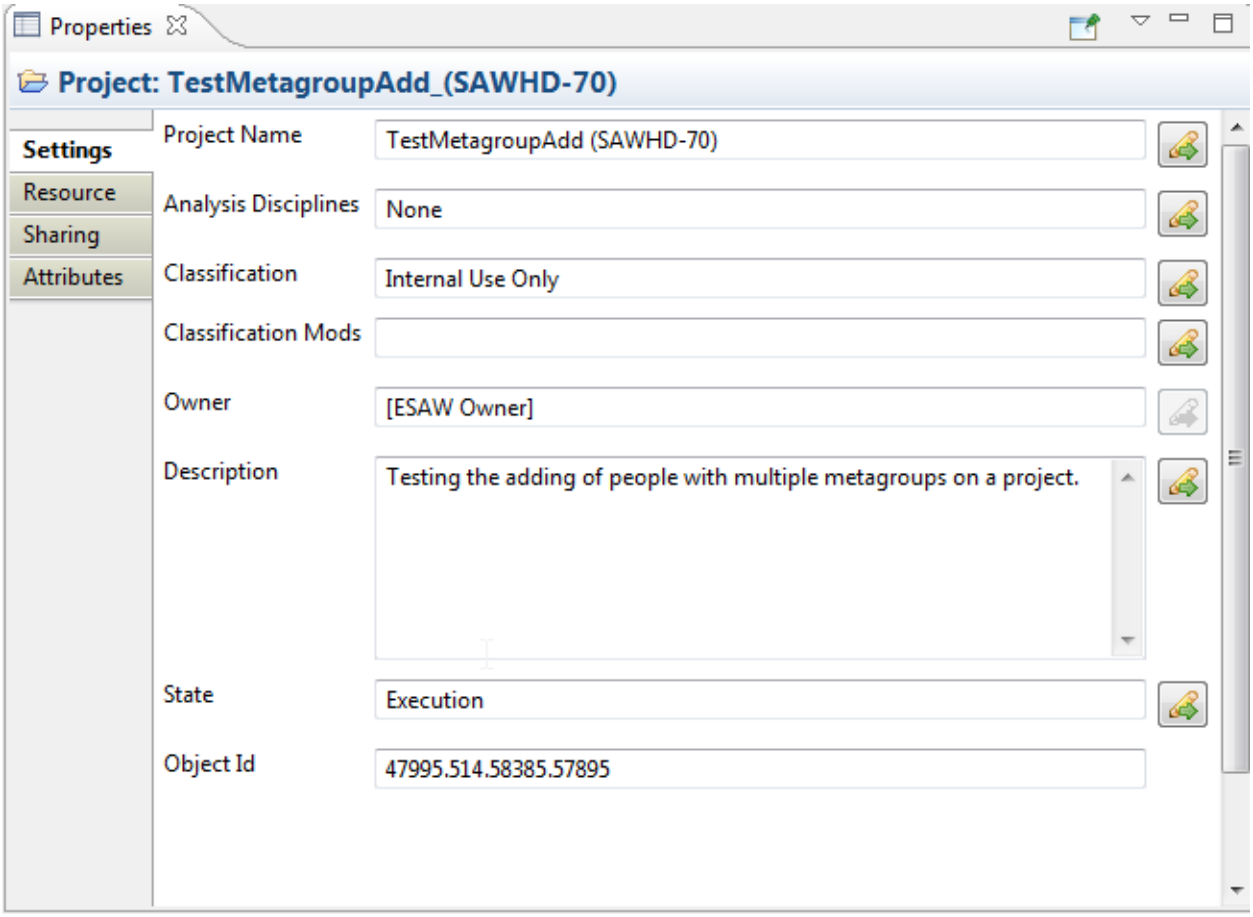

Figure 2. Project attributes

4.1.3 Advance Lifecycle

Projects have a defined lifecycle. During the Lifecycle, different access permissions are available. It is expected that different team members would participate at different points in the Project Lifecycle.

4.1.4 Modify Attributes

The attributes associated with a project may be modified by a Team Lead. Certain attributes may be restricted in the ability to be changed by Business Rules, such as lowering the classification of a project when it contains artifacts of a higher classification.

4.1.5 Adjust Team Members (see Team Actions)

4.2 Folder Actions

4.2.1 Create Folder

Any member of the team should be able to create a folder at any level in the project hierarchy. Folders are organizational containers that hold artifacts or other folders. The top level project is a specialized folder with additional attributes.

4.2.2 Add Artifact to Folder

Any Team Member may add an artifact to a folder in a given project. Artifacts may be linked across projects subject to NTK Business Rules.

4.2.3 Remove Folder from Folder

Any Team Member may remove a folder from a parent folder (subject to folder locking). How the recursive contents are handled depends upon Business Rules.

4.2.4 Remove Artifact from Folder

Any Team Member may remove an artifact from a folder. An artifact need not have a parent folder (though it will always have a controlling Project). In essence, removing an artifact from a folder is removing the relationship between the two. The removal of an artifact may be a precursor to the deletion of the artifact, but said deletion is not a requirement. In addition, since an artifact may "exist" (i.e., be linked to) multiple parent folders, removing an artifact from a folder does not imply deletion. Business Rules or a locked folder may prevent detaching an artifact from a folder.

4.2.5 Delete Folder

Any Team Member may delete a folder in a project. Business Rules determine how recursive contents are handled. Most specifically, some artifacts may be deleted (if they are not referenced elsewhere) and some may merely be detached from the folder prior to the folder being deleted.

4.2.6 Update Folder Attributes

The *owner* of a folder or any Team Lead may modify the attributes of a folder. Business Rules may prohibit certain modifications, such as raising a folder's classification higher than a parent folder. The "name" of a folder is considered an attribute.

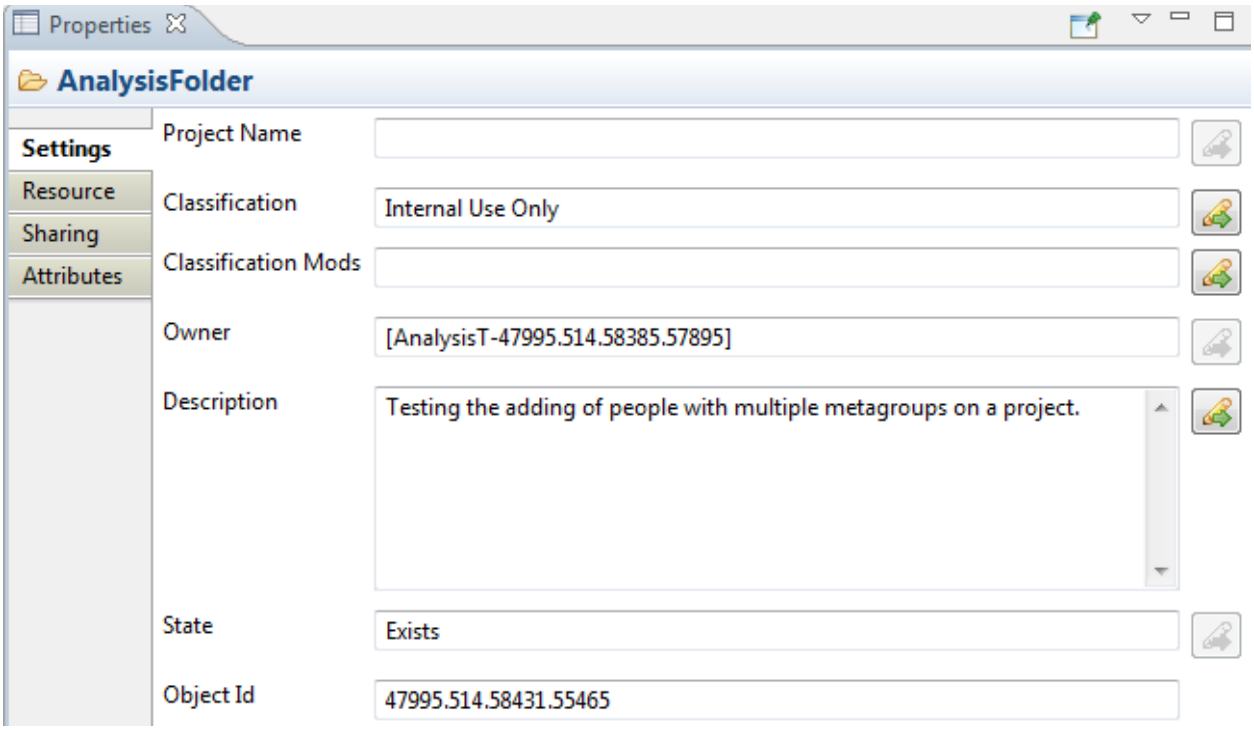

Figure 3. Folder attributes.

4.2.7 Adjust Sharing

Any Team Lead may adjust the sharing of a folder by applying a group to the folder. Such sharing is inherently an NTK operation. Artifacts already present in the folder *shall* have their visibility adjusted. New artifacts added to a folder *shall* have the visibility associated with the new scoping.

4.2.8 Register for Notifications

Any Team Member may register for notifications for changes to the folder. Changes include adding or removing children (other folders or artifacts).

4.3 Artifact Actions

4.3.1 Create Artifact

Any Team Member may create an artifact in a folder. An artifact is an object that represents the metadata about some deliverable. It may have *N* files attached to it, though current Business Rules limit a single file to an artifact. Any file may have *N* file versions attached. A file may be stored in the repository, or it may be a link to an external storage location. Artifacts may be of various types, and each type has a set of attributes associated with it. The specific attributes associated with the artifact may be modified by the *owner* of the artifact (usually the individual who created the artifact), or a Team Lead.

4.3.2 Add File Version to Artifact

Assuming the artifact is not locked to a specific individual, any Team Member may commit a new version of a file to an artifact.

4.3.3 Add Artifact to Folder

Within the *same* project, any Team Member may add an artifact to any project. An artifact may have multiple parent folders (i.e., it may be linked in multiple places), but exists as a single object. Adding an artifact to a folder in a *different* project may be restricted due to NTK Business Rules. When an artifact is added to a folder, Business Rules are processed to ensure compliance with certain situations. An example of a Business Rule is that an artifact cannot be added to a folder with a lower classification. When an artifact is added to a folder, Business Rules may automatically apply certain default settings (e.g., classification).

4.3.4 Update Artifact Attributes

The artifact *owner* or any Team Lead may modify any attribute of an artifact. Artifacts have a type, and each type has specific attributes.

4.3.5 Move Artifact to Another Folder

Any Team Member may move an artifact between folders within the same project. This capability is needed to support refactoring.

4.3.6 Modify Artifact Lifecycle State

Artifacts have an associated Lifecycle. The owner or a Team Lead may modify the Lifecycle state of an artifact. The operations permitted against an artifact may differ by lifecycle state. The states and access permissions are definable by the type of the artifact. Modifying the Lifecycle state collects an electronic signature and optional comments. Business Rules may prohibit certain transitions. Additional Business Rules may be invoked upon a transition (such as keeping only the latest file version).

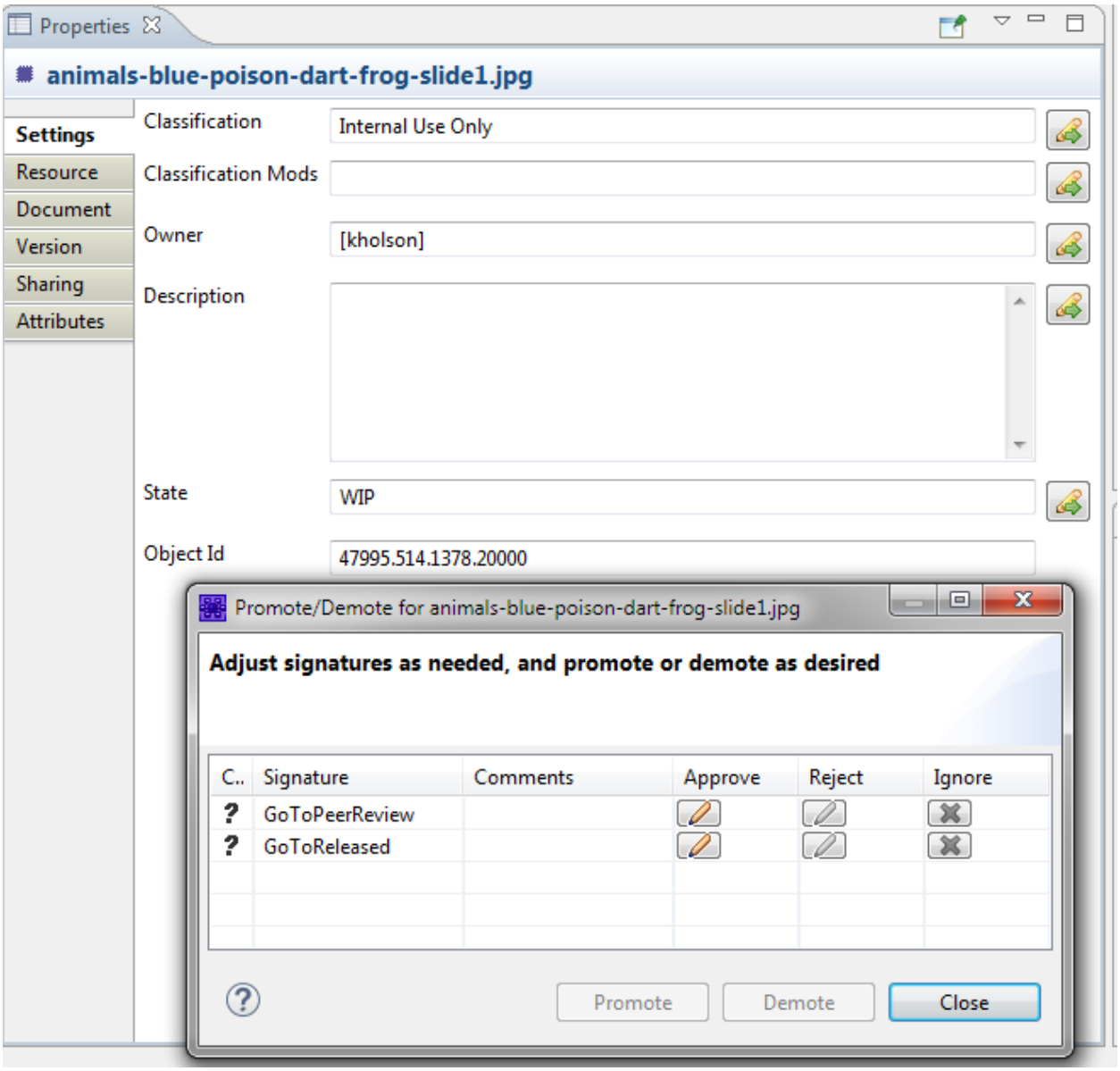

Figure 4. Updating lifecycle state for an artifact.

4.3.7 Delete File Version(s)

The artifact *owner* or any Team Lead may delete specific versions of a file attached to an artifact.

4.3.8 Delete Artifact

The artifact *owner* or any Team Lead may delete an artifact. Deleting an artifact removes it from all parent folders, removes all files and file versions from the file storage location (note: externally referenced files are not affected), and the artifact metadata is deleted. A history notation about an artifact deletion should be available.

4.3.9 Search for Artifact by Attributes

An artifact may be found by searching for values on any of its attributes. All search results are constrained *on the server* to match only authorized results. Note that searching by full text in a file attached to an artifact should also be supported, but the returned Artifacts must be filtered on the server as well.

4.3.10 Specify Access Control on an Artifact (e.g., group or role)

Artifacts, by default, are visible only within the context of a project, and thus limited to the team. Access may be reduced to only the Object Owner, or may be expanded by associating other groups with the artifact.

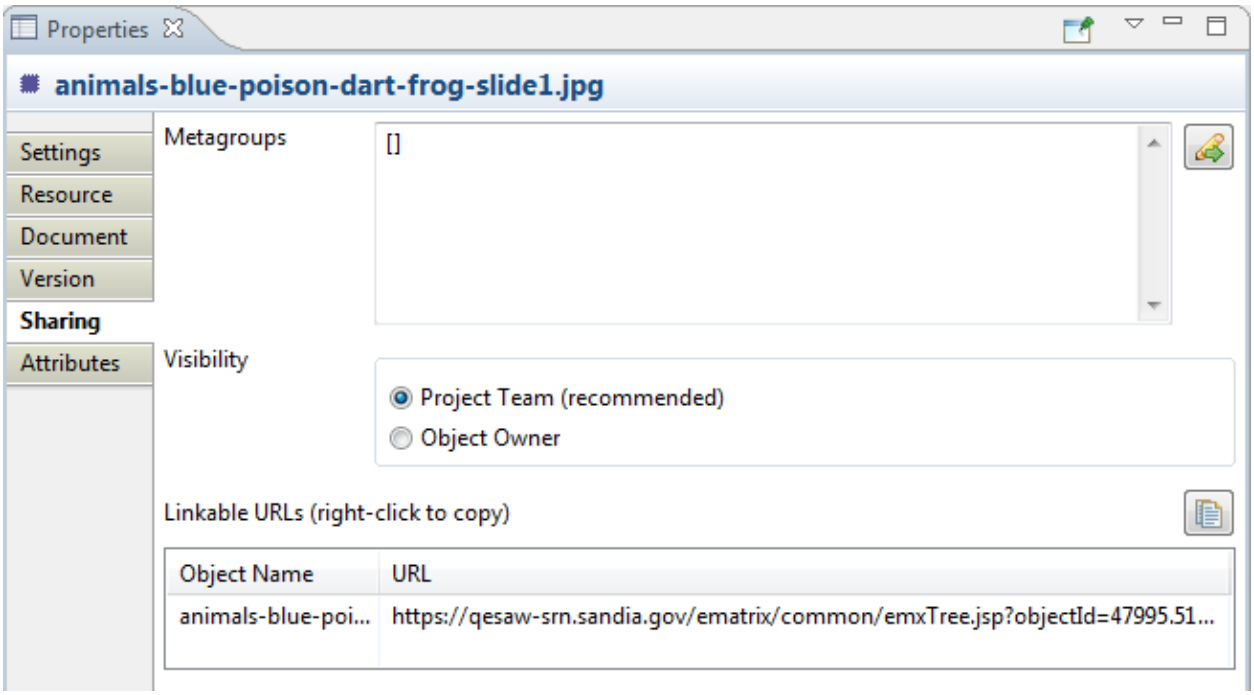

Figure 5. Specifying access control on an artifact.

4.3.11 Change Owner of an Artifact

The current *owner* or any Team Lead may change the ownership of an artifact. Artifact ownership has certain NTK responsibilities associated with it, and in some cases an owner may perform actions (such as locking for modification).

4.3.12 Relate Artifact to another Artifact

An artifact may be related to another artifact. For example, there may be an artifact type of "Engineering Note" that may be related an artifact type of "Analysis Artifact". The specific relationship type determines the cardinality and directionality of any relationship, and specific relationships may carry particular attributes as well. The state of an artifact may prevent certain associations, and other Business Rules may also be applicable.

4.3.13 Relate Artifact to other Object

There are instances where an artifact should be relatable to other objects, such as Requirements. The specific use cases vary slightly by the other Object's controlling lifecycle and attributes, as well as the relationship between the Artifact and the other object. Nonetheless, the general case is that any Team Member may associate an Artifact with another relevant object in the Project.

4.3.14 Lock to prevent modifications

An *owner* or a Team Lead may lock an artifact to prevent modifications. The attributes on a locked artifact may only be modified by the locker, and only the locker may add new file versions to the artifact. The locked status does not affect viewing the artifact or downloading files (subject to other access control issues). Any Team Lead may remove a lock on an artifact.

4.3.15 Register for Notifications

Any Team Member may register for notifications about changes to an artifact. One may register for notifications about new files or file versions added, if the artifact is added to a folder (moved or linked to an additional one), when it is locked or unlocked for modifications/file commits, or when attributes are modified.

4.4 Team Actions

4.4.1 Add Person to Team

Any Team Lead may add an individual to the project team. Adding a Team Member is inherently an NTK decision. Any registered user may be added to the team. Business Rules may prohibit certain additions depending upon project attributes (e.g., classification).

4.4.2 Change Role of Person on Team

Any Team Lead may modify the role of a person on the team. An individual may be changed from a Team Member to a Team Lead or the Project Manager. An individual may be changed from a Team Lead to a Team Member. Changing the role affects certain access permissions, as well as the ability to change team membership and participate in artifact sharing (a form of NTK).

4.4.3 Remove Person from Team

Any Team Lead may remove an Team Lead or Team Member from a Project. The Project Manager may not be directly removed from the team, but instead a new Project Manager must be assigned, and then the previous Project Manager (who will now be a Team Lead) may be removed.

4.5 History Actions

4.5.1 Automatic History

Most activities (e.g., adding an artifact, committing a file version to an artifact, modifying attributes, etc.) automatically generate a history entry. History entries must be relatable to the user, the time of the action, and the specific object.

4.5.2 Add User Defined History Entry

It must be possible to add User Defined history to an Object. That is, history entry types must not be limited only to system generated entries.

4.5.3 Obtain History for an Object

For any Object, it must be possible to obtain the history entries relating to that object.

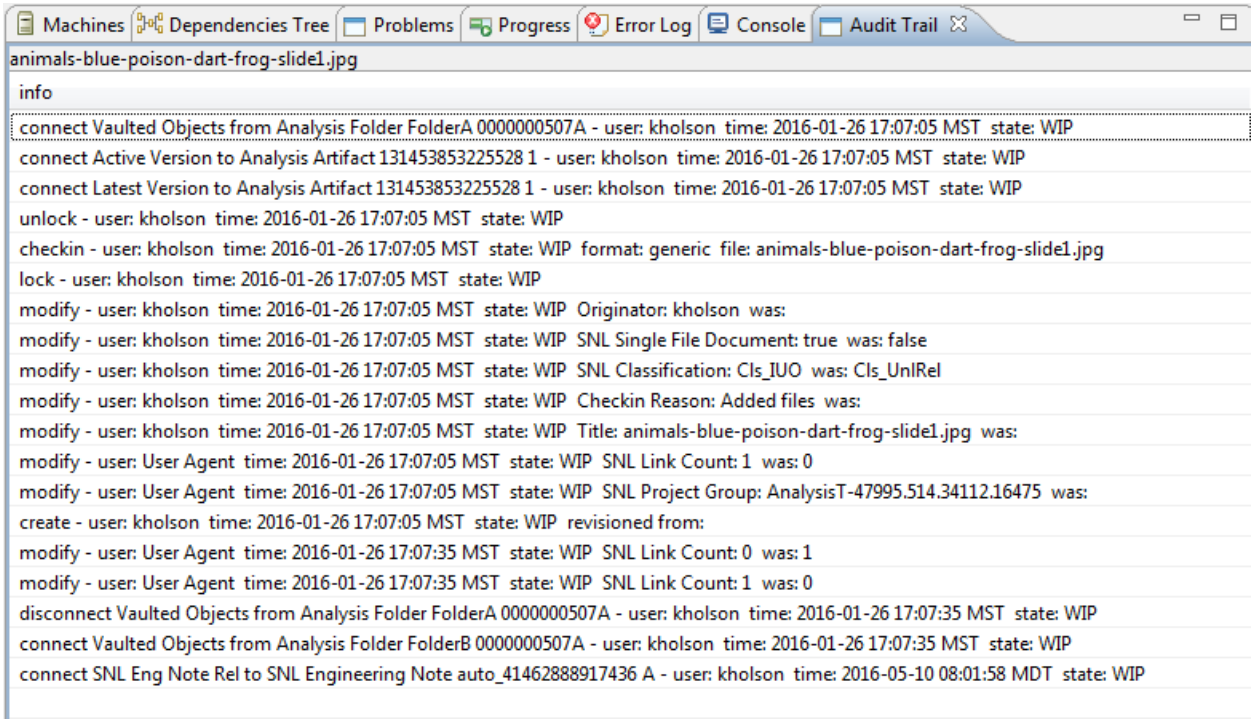

Figure 6. Obtaining history for an object.

5 TECHNICAL REQUIREMENTS

In this section, we provide a list of fundamental properties and capabilities that the proposed new system must support from a developer's perspective. In general, the intent is that operations listed here must be a primitive operation available on the system via an API, and each of these properties is essential to our ability to implement the new service.

5.1 Open Interface

5.1.1 Must be possible to integrate with Java, Python, command-line front ends

5.2 No Runtime Cost

5.2.1 Must be possible to deploy with no commercial license encumbrance

5.3 Transaction Management

5.3.1 Transaction support required

5.3.2 ACID compliance

5.3.3 Both read and update transactions

5.4 Ability to define Attributes

5.4.1 Type (e.g., String, float, etc.)

5.4.2 Default Values

5.5 Ability to define various "types" of objects

5.5.1 Collection of attributes

5.5.2 Hierarchical and inheritable

5.5.3 Optionally used in search

5.6 Ability to separate data into different shards or tablespaces

5.6.1 Based upon type or controlling workflow

5.7 File Storage

5.7.1 External to the metadata management

5.7.2 Must support:

5.7.2.1 Stored files

5.7.2.2 Externally referenced files

5.7.2.3 Large file size

5.7.2.4 Progress monitoring on file upload/download

5.7.3 File Versioning

5.7.4 A File Version must exist as an object that may be related (e.g., baselining)

5.8 Index-based Search

5.8.1 Attribute based for selecting metadata

5.8.2 Full file text search

5.8.3 Must respect the security model

5.9 Flexible Authentication mechanisms

5.10 Authorization must

5.10.1 Be object-level based

5.10.2 Support object attributes for authorization adjudication

5.10.3 Support custom modules for extending the authorization model

5.10.4 Must be handled on the server (i.e., never return to the client data that is inappropriate)

5.11 Access and Collections must include

5.11.1 Person

5.11.2 Group

5.11.3 Role (but RBAC alone is insufficient)

5.11.4 Must be able to prevent "owner" from having access

5.11.4.1 If the owner loses an access capability (e.g, Sigma-15), then ownership alone must not allow access

5.12 Extensible Trigger System

5.12.1 Type based

5.12.2 Event driven for multiple event types (pre/override/post on various events such as create/update attribute/delete/etc.)

5.13 Ability to "subscribe" to various object events

5.13.1 Notification when certain object events occur, such as adding a file, changing the owner, etc.

5.14 Lifecycle states

5.14.1 Ability to define different lifecycles

5.14.2 Apply these lifecycles to the objects

5.14.3 Access control must be tied to lifecycle state

5.15 Administrative Console

5.16 Unique Object Reference (i.e., an Object Identifier)

5.16.1 Must be stable

5.16.2 Must not be tied to a display attribute that can change

5.17 History entries created automatically by various activities

5.17.1 Ability to add custom history entries

5.17.2 History must tie to specific objects

5.18 Relationships between objects

5.18.1 Attributes on relationships

5.18.2 Cardinality

5.18.3 Type limited

5.18.4 Direction (to/from based upon object type)

5.18.5 Ability to use in traversal

5.18.6 Participate in the Trigger system

5.19 Ability to lock an object to prevent updates and/or file additions

5.20 Robust Permission Model

5.20.1 Create

5.20.2 Read Attributes

5.20.3 Update Attributes

5.20.4 Delete

5.20.5 Relate to a Parent Object

5.20.6 Relate to a Child Object

5.20.7 Remove Relationship

5.20.8 Add File Version

5.20.9 Remove File Version

5.20.10 Download File Version

5.20.11 Create Copy

6 CASE STUDY

6.1 The Sandia Analysis Workbench (SAW)

Sandia's SAW product uses a classic client-server architecture (see Figure 7.) Both client and server are portable and written in Java. The client is built on the Eclipse platform[†], an application framework that allows for modular development of complex, extensible applications. The SDM client in SAW is all custom code.

The SAW SDM server uses a web application architecture. The server is deployed in the Tomcat application container[‡]. Client-server communications are implemented using Spring Remoting[§], using HTTPS as the transport protocol – meaning that the SDM server essentially uses "web services" as its API. Authentication is handled using Sandia's Kerberos infrastructure.

The object store is based on eMatrix, a commercial PDM product which is simply a Java library that forms part of the server application. Usage of eMatrix itself is therefore entirely within the server. eMatrix in turn stores its data in the corporate Oracle database. The filestore is currently located on Sandia's corporate SAN.

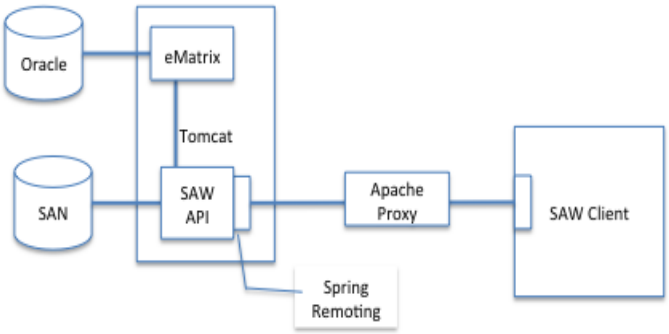

Figure 7: SAW SDM Architecture

6.2 Current Risks and Limitations

The immediate motivation for replacing the SAW SDM service is to reduce risk to existing deployments, and to ease adoption for new customers by reducing the expense of standing up a central repository. The SAW SDM server is based on an old version of the commercial eMatrix product, and Sandia is no longer paying for maintenance. Significant changes to the software would be necessary to move to a newer version of eMatrix, and it's not clear that the necessary features are available in newer versions. If this older version of eMatrix is incompatible with future changes to the Java Virtual Machine or the Linux OS, the SAW SDM server would be tied

---

[†] http://www.eclipse.org

[‡] https://tomcat.apache.org

[§] http://docs.spring.io/spring/docs/current/spring-framework-reference/html/remoting.html

to obsolete versions of those elements as well, and Sandia security mandates would eventually make it impossible to continue without updating or replacing eMatrix.

Faced with the choice between paying annually to maintain an eMatrix license – and still needing to do substantial work to modify the server to work with newer software – or abandoning eMatrix altogether, the latter seems like a more cost-effective approach in the long run. Furthermore, even if we were to reduce risk by upgrading, we would still face the issue that expensive third-party licenses impede adoption by outside customers. Therefore, the decision to leave eMatrix in favor of an unencumbered alternative seems obvious.

The current SAW SDM architecture is based around the idea of data physically residing on a filestore associated with the repository. Files are uploaded to and downloaded from the filestore via the server's web services API. Although the current server easily handles multi-gigabyte files, it is not designed to store large numbers of such files, nor can it handle larger files in the terascale or larger regimes that cannot be easily moved.

The current server supports the notion of a file on an external data store, although this concept isn't exposed to the user. Ideally, in the next generation server, large files would be handled by such first-class references to external data, so files residing on other systems could be easily referenced *in situ*. Besides improving scalability, this would allow the new SDM server to federate with other data management systems.

7 SOFTWARE SPECIFICATION

7.1 SDM Object Model

Our SDM data model consists basically of four data types:

- Container
- Document
- Relationship
- Administrative Objects (Person, Role, Group)

Figure 7 shows a simplified version of an SDM object model that is created from these four data types. It is illustrative of the complexity of E-R modeling appropriate for our domain. As described in the text, it is richer than the model for a plain document management system, but less complex than the model used by a full product data management system.

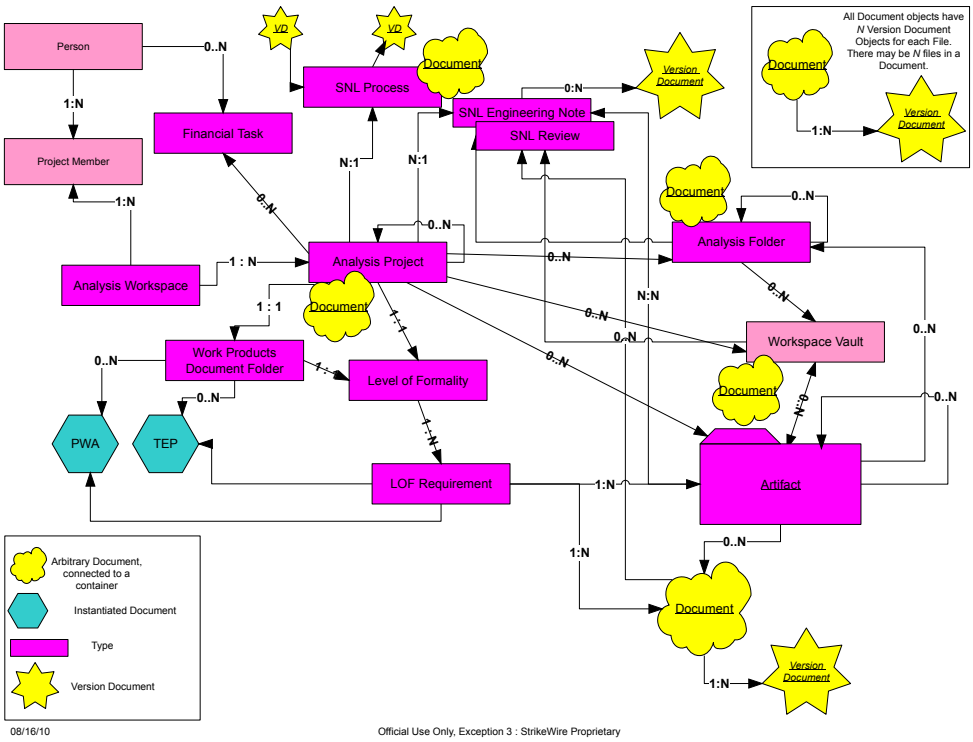

Figure 8. SDM Object Model

7.1.1 Container

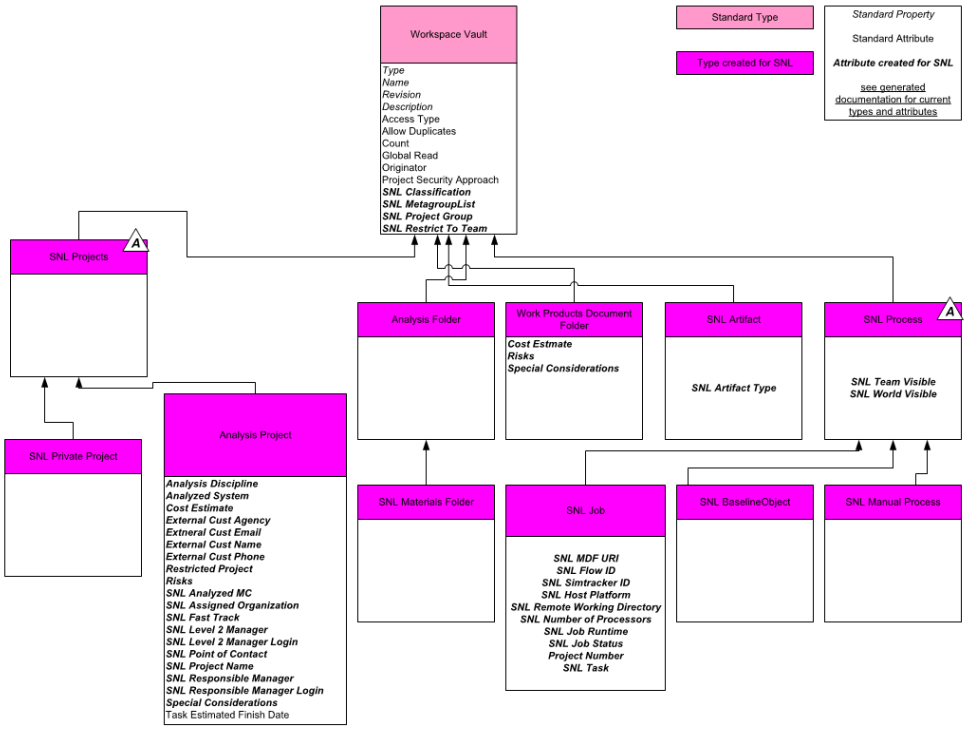

Figure 9: Container object model

7.1.2 Document

Document objects store essential metadata about files stored in the repository. They have version objects that store metadata about a specific file version. The actual files are associated with a version object.

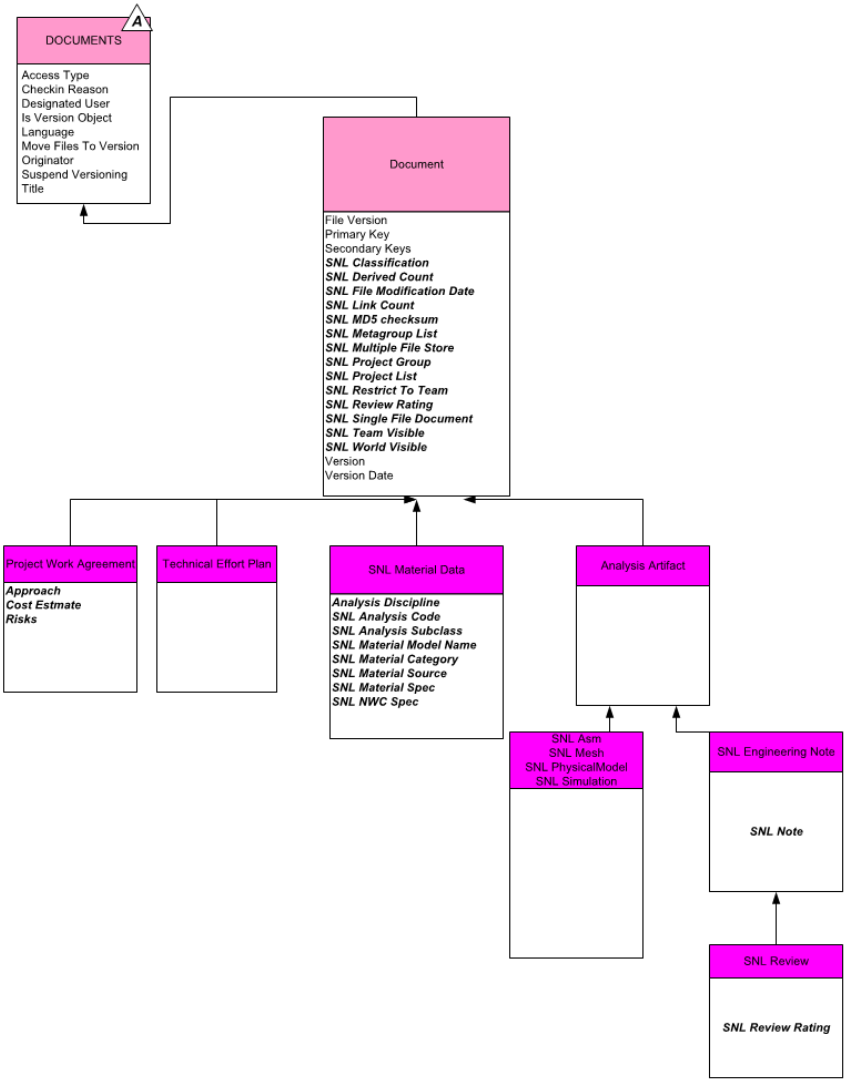

Figure 10: Document type object model

7.1.3 Relationships

Relationships have specific qualities such as type restrictions, direction, cardinality, etc.

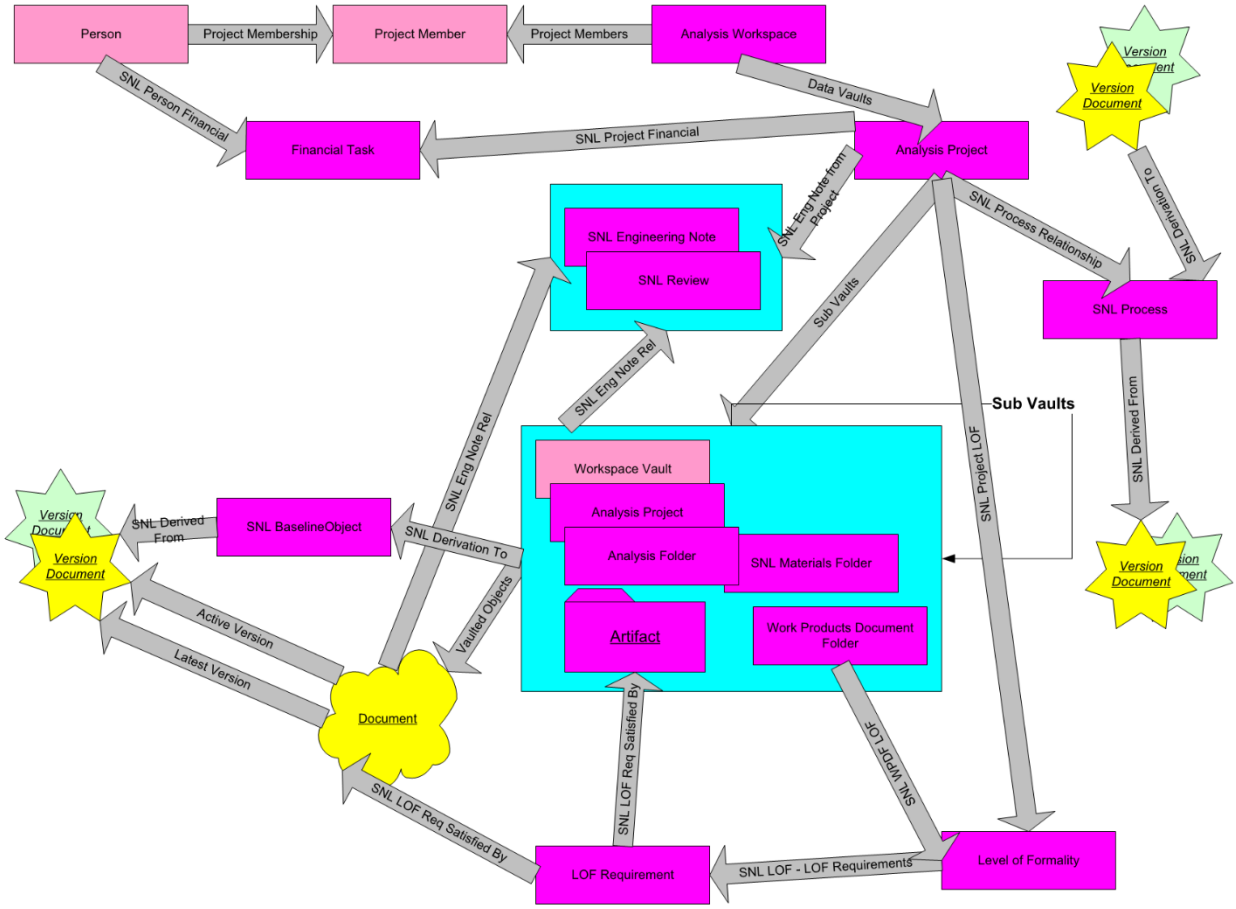

Figure 11. SDM Object Model

7.2 Proposed Implementation

In considering how to store simulation data and especially metadata, we have vetted several different approaches. For the purposes of narrowing the field, the following guidelines were utilized. An object here is a representation of some abstract manifestation of a concept found in the data model, such as a "document" or "folder."

- An ability to define "types" in an inheritable, hierarchical manner, where a type defines a series of expected attributes.
- The ability to model complex relationships between the objects, including attributes on the relationships.
- A powerful query system allowing for traversal of the relationships.
- An ability to integrate a robust security model leveraging attributes of an individual object.
- Open source
- Performant
- Transaction support
- Trigger Support
- Unique Identifier per record

While traditional relational databases such as MySQL could be utilized, using one creates a need to generate mappings between the programming objects and the underlying table-based stores. Traditionally, some type of Object Relational Mapping (ORM) system has been utilized. However, in our experience, ORM – while powerful – is also problematic in terms of performance and comprehensibility of the underlying data storage structure.

In the past decade, a new approach to data storage has arisen with the NoSQL movement. This approach emphasizes different storage approaches rather than the traditional table-based approach with rows and defined columns. NoSQL approach emphasizes a more flexible way to define an object, typically by leveraging key-value pairings as the definitional unit. In addition, relationship modeling with graph-based engines in another feature of several NoSQL databases.

After looking at several potential candidates, we propose to utilize OrientDB (http://orientdb.com/) as the underlying storage engine. OrientDB meets most of our previously-stated technical requirements:

- Documents, with schema-based typing, as a primary object
- A graph database that allows for efficient, performant queries
    - Documents are stored as a Vertex on the graph
    - Relationships are the Edges between the vertices
- Full text Search support
- ACID Transactions
- A robust security model, with triggers that allow for additional validation

- Open Source License

OrientDB also supports a variety of APIs to access the data, including Java, Python, PHP, and Perl. A webserver can be easily written to overlay these approaches to support, for example, a full RESTful approach.

The requirements outlined in this document could of course be met by multiple means. The largest decision would be between adopting and customizing an existing product versus creating a new product tailored to these requirements. In the past, we have surveyed the market of existing PDM software in an attempt to locate any tool that would meet our requirements, and have found none suitable. In industry, comparable systems are generally purpose-built using Java Servlet technology directly on object-relational mapping platforms like JDO[**] or JPA[††], or on NoSQL[‡‡] databases. A vast array of libraries and tools is available in this space for creating applications like our SDM service, which would present no special difficulties. Using these industry-standard tools and applicable best practices for architecture would allow us to leverage a great deal of existing work and provide a viable path to creating a sustainable system.

[**] https://db.apache.org/jdo/

[††] http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

[‡‡] https://en.wikipedia.org/wiki/NoSQL

8 APPENDIX I: GLOSSARY

Term	Definition	Comments
ACID	Atomic, Consistent, Isolated, Durable.	
API	Application Programmer's Interface	
Application Layer	layer 2: a simulation code or similar composition of packages, interacts across memory hierarchies and archival storage and data management systems. For a simulation code the subcomponents model physics packages and their interaction with analysis and I/O. This layer is the purview of the code developer and third party library developer, as well as systems software etc.	
Artifact	An artifact is an object that represents the metadata about some deliverable. It may have N files attached to it, though current Business Rules limit a single file to an artifact. Any file may have N file versions attached. A file may be stored in the repository, or it may be a link to an external storage location. Artifacts may be of various types, and each type has a set of attributes associated with it. The specific attributes associated with the artifact may be modified by the owner of the artifact (usually the individual who created the artifact), or a Team Lead.	
Baseline	A baseline is a collection of documents that go together to form a higher level object. For example, baselines are used in design engineering to tag part files that make up an assembly. In modeling and simulation, baselines can be used to keep track of the files associated with a model and its simulation results files. (Sandia definition, seems to conflict with LLNL usage of this term)	
Business Rules	Business rules are statements that tell you whether you may or may not do something, or give you the criteria and conditions for making a decision. They can be implemented as site-specific customization of general processes.	
CAD	Computer-Assisted Drafting	
Campaign Layer	a process over time of repeated process layer jobs, as user(s) change approach, physics, in order to complete a campaign or project.	
CompSim	Computational Simulation	

Data	numerical and/or categorical information. Data may be used as input to simulations (e.g., equation of state tables), generated as output (time histories, Silo files), produced in digest form (statistical summaries, ultra files), contained in log files in human readable form (e.g., time step information, indices and locations of zones with negative sub-volumes) The size of data used as input and generated as output is widely varying, so we probably need a nomenclature for distinguishing data that might be amenable to longer term storage and index, vs. data that needs to reside on a parallel filesystem.
Digest	Data of smaller size that describe the simulation either in aggregate, or in a sample-based fashion. Examples include ultra curves, scalar figures of merit (integrated values over time and/or space), movies, reduced models (e.g., statistical information, morse theoretical structures such as contour trees), images (e.g., simulated diagnostics), ...
Document	A human readable form of information that may or may not contain data (most likely in digest form) and may be considered a discrete entity. Examples: Log files, power point, PDF files, human generated simulation input decks/scripts, post-processing scripts. Documents will often have relationships to data and simulations (data model to be determined!)
File	a collection of bytes stored in a filesystem, object store, or content addressable store (like git). A single data set might be composed of many files. Documents are typically one file, although a LateX 'document' is often built out of many source files.
Layer	a hierarchically defined subset of a Simulation workflow, layer 0 being the outer most layer (see below) encompassing the interface to the user(s), with lower/deeper layers increasingly defined by computer science and system engineering considerations (e.g., in-situ coupling)
LANL	Los Alamos National Laboratories
LLNL	Lawrence Livermore National Laboratories
Meta-data	Data about a simulation, including identifiers the uniquely specify inputs that might be hosted elsewhere, file paths, physics and other parameters.

	Meta-data may be a function of time (e.g., mesh management parameters for ALE).
Model	The set of data and documents that define a simulation. This includes input decks, meshes, data files, etc.. A model can be composed of several baselines under the Sandia definition. Example: starting from a Shock tube problem definition, the model would consist of the mesh specification or input script, the material data (densities etc.), equation of state selection, numerical solution methods and approaches, and other aspects of setting up a simulation code.
Model (SNL definition)	The set of data and documents that specify a particular simulation of a problem. That is, mesh descriptions, material data and other tabular or other specifications, simulation input scripting and settings, job submission requirements (number of nodes and domains etc).
NTK	Need To Know
Package Layer	layer 3: the processing of kernels and their interaction with compute hardware, including cache levels, memory hierarchies. In this naming convention, a task based parallel infrastructure might be considered a package layer workflow engine. Alternatively, this could be the per-core aspect of a simulation workflow
Parameter Study	
PDM	Product Data Manager
Problem	The set of data and documents describing the physical characteristics of an experiment that, when coupled with a model, can be used to produce simulations of the experiment. Example: a shock tube problem might consist of the physical dimensions of the tube and membrane, identify the materials involved, but not the characteristics of those materials, and the characteristics of the shock generation.
Process Layer	this layer interfaces with the user ('a person who runs codes'), may involve the running of suites of applications, or sets of closely coupled applications (in the sense that the user must manage inputs and outputs and execution of some simulations and inputs to others). Defines and end-to-end repeatable process (e.g.,

	scriptable), where the user interacts with HPC environment, constructs code inputs, and schedules/runs analysis applications.	
Project (SNL)	Projects associate a given body of work within an hierarchical structure with a given set of team members. Projects are intended to have a specific lifetime, and a project should have an identifiable set of deliverables. Of course, projects may be used as a general storage area without the need for a time frame or specific outcome. By associating teams to projects, a project is a primary focus for access control. Roles: • team member: may create folders and artifacts in a projects • team lead: is team member, and may adjust team membership and add meta-groups • project lead: is team lead, and may make changes pertaining to lifecycle of the project	
RBAC	Role-Based Access Control	
Run	A single instance of one or more simulation codes and the output for it, e.g. 1 msub / sbatch submission. A problem combined with a model may be executed and thus generate several runs.	clarify
SDM	Simulation Data Management	
Simulation Workflow	The process of running an simulation or simulations, which may or may not be closely coupled, in the service of a WCI Workflow. In LANL parlance a workflow (and its layers) can be separated into distinct phases.	
SNL	Sandia National Laboratories	
Step	A single instance of something that can be executed. This does not need to be a simulation. e.g. a single `srun` call, calling a script	clarify
Study	The collection of runs whose results will be observed together.	clarify
Suite	A collection of studies, which may include re-runs, and may involve multiple different simulation codes	clarify
Workflow (generic)	an orchestrated and repeatable pattern of business activity enabled by the systematic organization of resources into processes that transform materials,	Wikipedia definition

	provide services, or process information
Workflow (WCI)	the process of simulation, experiment, and theory, by single individuals or work groups, required to complete a campaign or project and propagate/disseminate its results

DISTRIBUTION

[List external recipient names and addresses]

4 Lawrence Livermore National Laboratory
Attn: N. Dunipace (1)
P.O. Box 808, MS L-795
Livermore, CA 94551-0808

[List in order of lower to higher Mail Stop numbers.]

1	MSXXXX	Name of Person	Org. Number
1	MSXXXX	Name of Person	Org. Number
1	MS0899	Technical Library	9536 (electronic copy)

Sandia National Laboratories